STRINGS

String: It is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.

Ways of Creating a String

There are two ways to create a string in Java:

- String Literal
- Using new Keyword

String literal
String s = “GeeksforGeeks”;

Using new keyword
String s = new String (“GeeksforGeeks”);

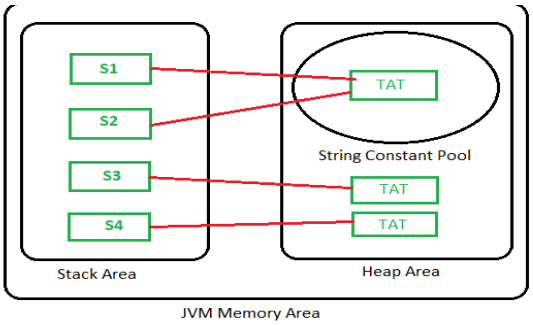
StringBuffer is a peer class of String that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Syntax:
StringBuffer s = new StringBuffer("GeeksforGeeks");

StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters.

Syntax:
StringBuilder str = new StringBuilder();
str.append("GFG");

```
class StringStorage {
        public static void main(String args[])
        {
                String s1 = "TAT";
                String s2 = "TAT";
                String s3 = new String("TAT");
                String s4 = new String("TAT");
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
                System.out.println(s4);
        }
}
```



Note: All objects in Java are stored in a heap. The reference variable is to the object stored in the stack area or they can be contained in other objects which puts them in the heap area also.

Immutable String in Java

- In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.
- Once a string object is created its data or state can't be changed but a new string object is created.

// Java Program to demonstrate Immutable String in Java

```
import java.io.*;
class GFG {
	public static void main(String[] args)
	{
		String s = "Sachin";

		// concat() method appends
		// the string at the end
		s.concat(" Tendulkar");
		// This will print Sachin
		// because strings are
		// immutable objects
		System.out.println(s);
	}
}
```

Here Sachin is not changed but a new object is created with "Sachin Tendulkar". That is why a string is known as immutable.
As you can see in the given figure that two objects are created but s reference variable still refers to "Sachin" and not to "Sachin Tendulkar".

```
// Java Program to demonstrate Explicitly assigned strings
import java.io.*;

class GFG {
	public static void main(String[] args)
	{
		String s = "Sachin";
		s = s.concat(" Tendulkar");
		System.out.println(s);
	}
}
```

Why are string objects immutable in java?
Because Java uses the concept of string literal. Suppose there are 5 reference variables, all referring to one object “sachin”. If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

String(byte[] byte_arr) – Construct a new String by decoding the byte array. It uses the platform’s default character set for decoding.
Example:
byte[] b_arr = {71, 101, 101, 107, 115};

String s_byte =new String(b_arr); //Geeks

String(byte[] byte_arr, int start_index, int length) – Construct a new string from the bytes array depending on the start_index(Starting location) and length(number of characters from starting location).
byte[] b_arr = {71, 101, 101, 107, 115};
String s = new String(b_arr, 1, 3); // eek

String(char[] char_arr) – Allocates a new String from the given Character array
Example:
char char_arr[] = {'G', 'e', 'e', 'k', 's'};
String s = new String(char_arr); //Geeks

String(StringBuffer s_buffer) – Allocates a new string from the string in s_buffer
Example:
StringBuffer s_buffer = new StringBuffer("Geeks");
String s = new String(s_buffer); //Geeks

String(StringBuilder s_builder) – Allocates a new string from the string in s_builder
Example:
StringBuilder s_builder = new StringBuilder("Geeks");
String s = new String(s_builder); //Geeks

String Methods

1. int length(): Returns the number of characters in the String.
   "GeeksforGeeks".length(); // returns 13
2. Char charAt(int i): Returns the character at i th index.
   "GeeksforGeeks".charAt(3); // returns ‘k’
3. String substring (int i): Return the substring from the ith index character to end.
   "GeeksforGeeks".substring(3); // returns “ksforGeeks”
4. String substring (int i, int j): Returns the substring from i to j-1 index.
   "GeeksforGeeks".substring(2, 5); // returns “eks”
5. String concat(String str): Concatenates specified string to the end of this string.
   String s1 = ”Geeks”;

String s2 = ”forGeeks”;
String output = s1.concat(s2); // returns “GeeksforGeeks”

6.

int indexOf (String s): Returns the index within the string of the first occurrence of the specified string.

String s = ”Learn Share Learn”;
int output = s.indexOf(“Share”); // returns 6

7.

int indexOf (String s, int i): Returns the index within the string of the first occurrence of the specified string, starting at the specified index.

String s = ”Learn Share Learn”;
int output = s.indexOf("ea",3);// returns 13

8.

Int lastIndexOf(String s): Returns the index within the string of the last occurrence of the specified string.

String s = ”Learn Share Learn”;
int output = s.lastIndexOf("a"); // returns 14

9.

boolean equals(Object otherObj): Compares this string to the specified object.

Boolean out = “Geeks”.equals(“Geeks”); // returns true
Boolean out = “Geeks”.equals(“geeks”); // returns false

10.

boolean equalsIgnoreCase (String anotherString): Compares string to another string, ignoring case considerations.

Boolean out= “Geeks”.equalsIgnoreCase(“Geeks”); // returns true

11. Boolean out = “Geeks”.equalsIgnoreCase(“geeks”); // returns true

int compareTo(String anotherString): Compares two strings lexicographically.

int out = s1.compareTo(s2); // where s1 and s2 are
// strings to be compared

This returns the difference s1-s2. If :
out < 0 // s1 comes before s2
out = 0 // s1 and s2 are equal.
out > 0 // s1 comes after s2.

12.

int compareToIgnoreCase(String anotherString): Compares two strings lexicographically, ignoring case considerations.

int out = s1.compareToIgnoreCase(s2);
// where s1 and s2 are
// strings to be compared

This returns the difference s1-s2. If :
out < 0 // s1 comes before s2
out = 0 // s1 and s2 are equal.

out > 0 // s1 comes after s2.

13. Note- In this case, it will not consider the case of a letter (it will ignore whether it is uppercase or lowercase).

String toLowerCase(): Converts all the characters in the String to lowercase.

```
String word1 = “HeLLo”;
String word3 = word1.toLowerCase(); // returns “hello"
```

14. String toUpperCase(): Converts all the characters in the String to uppercase.

```
String word1 = “HeLLo”;
String word2 = word1.toUpperCase(); // returns “HELLO”
```

15. String trim(): Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.

```
String word1 = “ Learn Share Learn “;
String word2 = word1.trim(); // returns “Learn Share Learn”
```

16. String replace (char oldChar, char newChar): Returns new string by replacing all occurrences of oldChar with newChar.

```
String s1 = “feeksforfeeks“;
String s2 = “feeksforfeeks”.replace(‘f’ ,’g’); // returns “geeksgorgeeks”
```

17. Note:- s1 is still feeksforfeeks and s2 is geeksgorgeeks